Semester 2 Examinations 2018/2019

Exam Code(s)	4BCT, 4BP, 4BLE
Exam(s)	B.Sc. in Comp Sc. & Information Technology B.E. in Electronic & Computer Engineering B.E. in Electrical & Electronic Engineering
Module Code(s)	CT420
Module(s)	Real-Time Systems
Paper No.	1
Repeat Paper	N
External Examiner(s)	Dr. Jacob Howe
Internal Examiner(s)	Prof. Michael Madden *Dr. Michael Schukat

***Instructions*:** Answer any 3 questions.
All questions carry equal marks.

Duration	2 hours
No. of Pages	7
Discipline	Information Technology

Requirements:
MCQ
Handout
Statistical/ Log Tables
Cambridge Tables
Graph Paper
Log Graph Paper
Other Materials

Release to Library: Yes [✓ X] No [ ]

Q1

(i) Using the code fragment below, explain the problem of **non-re-entrant code**:

```
int t;

void swap(int *x, int *y)
{
 t = *x;
 *x = *y;
 // hardware interrupt might invoke isr() here!
 *y = t;
}

void isr()
{
 int x = 1, y = 2;
 swap(&x, &y);
}
```

[10]

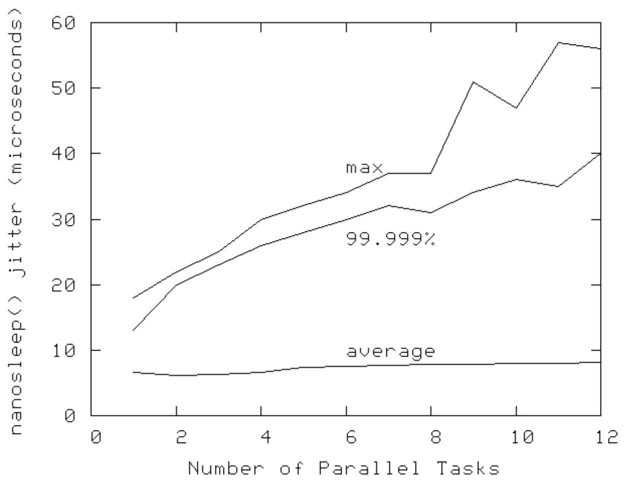
(ii) Distinguish between **Media Independent** and **Media Dependent Forward Error Correction** (FEC) strategies for Voice over IP, using examples to illustrate your answer. Comment also on their respective impact on delay, bandwidth utilisation and quality.

[10]

(iii) Distinguish between **RAID-4, RAID-5** and **RAID-6**, commenting on their characteristics, similarities, differences and I/O performance.

[12]

(iv) Explain how the Posix.4 call **nanosleep(timespec *a, timespec *b)** works and comment on the diagram below:



[8]

PTO

Q2

(i) Construct a **cyclic executive (CE)** schedule for the set of tasks listed in the table below. Furthermore, draw the **CE timeline** and determine how much CPU time is left for the processing of asynchronous interrupts.

Task	Period (msec)	Execution Time (msec)
A	10	2
B	10	2
C	20	1
D	20	3
E	40	2

[8]

(ii) Explain the inner workings of the **NTP** time synchronisation protocol. In your answer outline the calculations for **Round Trip Delay (RTD)** and **Offset**, while also summarising the different filter operations applied.

[12]

(iii) Using examples explain the **Hamming (7, 4) code with even parity**, highlighting its error detection and error correction capabilities.

[10]

(iv) Explain how you would estimate the **worst case execution time** of the code fragment below. In your answer make reference to **loop bounds** and identify the (in your opinion) **longest path**.

```
     const int max = 100;
     foo (float x) {
A:    for(i = 1; i <= max; i++) {
B:      if (x > 5)
C:        x = x * 2;
        else
D:        x = x + 2;
E:      if (x < 0)
F:        b[i] = a[i];
G:     x = x * 2;
      }}
```

[10]

PTO

Q3

(i) Using examples, provide definitions / explanations for the following terms:

a.	Response time of a system	[1]
b.	Synchronous tasks versus asynchronous tasks	[2]
c.	Slow, fast, soft and hard RTS	[4]
d.	Event	[1]
e.	Deep and shallow embedded systems	[2]
f.	Special function register	[2]
g.	Bathtub curve (as used in reliability engineering)	[3]
h.	Call stack	[5]

(ii) What is meant by a **volatile** variable in the programming language C? Use code fragments or diagrams to explain when such variables are needed.
[8]

(iii) Briefly discuss why networked systems (where packets are routed through a subnet) are usually not considered to be hard RTS. Use diagrams to illustrate your answer.
[5]

(iv) Contrast and compare **N-modular hardware redundancy using a single voter** versus **N-modular hardware redundancy using a voter hierarchy**. Use diagrams to support your answer.
[7]

<u>PTO</u>

Q4

(i) Distinguish between the following voter types:

a. Formalised majority voter (FMV)
b. Generalised median voter (GMV)
c. Formalised plurality voter (FPV)
d. Weighted averaging (WA)

[8]

(ii) Apply FMV, GMV, FPV and WA to the following data set (consisting of 14 weight-value pairs) and determine the output of each voter:

Tag	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14
Wgt	0.1	0.1	0.05	0.15	0.05	0.05	0.05	0.04	0.06	0.05	0.05	0.15	0.05	0.05
Val	10	11	12	11	8	9	12	14	13	17	13	9	12	8

[8]

(iii) Using code snippets show how a dynamic software redundancy approach can be emulated in Java using exception handling.

[12]

(iv) Memory locking is an important feature of POSIX.4. Explain the code snippet below and use the 7-state process model in conjunction with an example to outline why memory locking is important for (hard) real-time systems. Further on, discuss the consequences of memory locking being applied too generously.

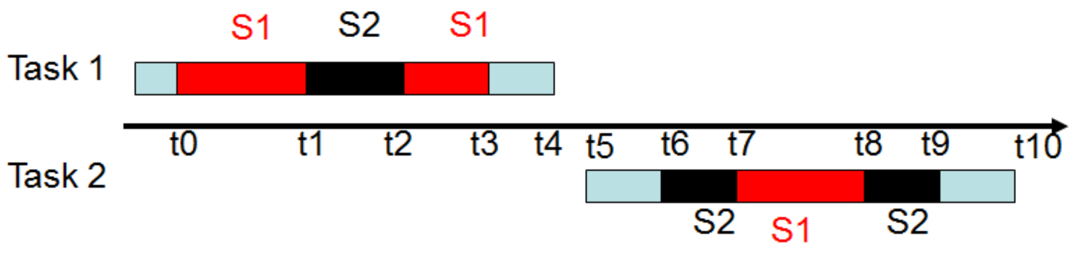
```
/*Main routine */
int main(){
/* Lock all process down */

mlockall(MCL_CURRENT|MCL_FUTURE);

… process code

munlockall();
return 0;
}
```

[12]

PTO

Q5

(i) The diagram below shows the timeline of two tasks, i.e. Task 1 and Task 2, which require at different times two semaphores, i.e. S1 and S2, in order to gain access to some critical resources. Note that Task 1 has a higher priority than Task 2. Modify the timeline to enforce a deadlock, and explain the sequence of steps that lead to this deadlock.



[5]

(ii) Using diagrams and code fragments, distinguish between the following two static software redundancy approaches:

a. N self-checking Programming using Acceptance Test

b. N self-checking Programming using Comparisons

[10]

(iii) The data below shows the output of the ntpq utility from 2 NTP clients, namely ClientA and ClientB. Based on the output, which client is most likely to deliver better synchronisation? Your answer should comment on redundancy, offsets, delay, stratum level and reachability.

ClientA	**RefID**	**st**	**t**	**When**	**Poll**	**Reach**	**Delay**	**Offset**	**Jitter**
+server1	serverx	2	u	51	64	156	391.281	6.24	7.79
-server2	servery	2	u	49	64	372	353.217	10.435	1.663
*server3	serverz	2	u	50	64	356	85.688	2.465	2.666
+server4	.PPS.	1	u	49	64	357	66.369	1.858	2.105
ClientB	**RefID**	**st**	**t**	**When**	**Poll**	**Reach**	**Delay**	**Offset**	**Jitter**
+server5	serverm	2	u	45	64	377	36.345	2.24	3.458
-server6	servern	2	u	42	64	377	313.217	3.435	1.11
*server7	.PPS.	1	u	13	64	377	10.688	1.485	1.455
+server8	.PPS.	1	u	22	64	377	28.456	1.818	1.345
+server9	.GPS.	1	u	18	64	356	15.345	-2.654	2.34

[15]

PTO

(iv) The (pseudo) code fragment below sets an interval timer using the POSIX API. Briefly explain the code and outline how process pre-emption could cause problems for the execution of this code.

```
clock_gettime(CLOCK_REALTIME, &now);
// Calculate interval (simplified):
Interval = $t_{abs}$ - now
// Create and set Interval timer:
timer_t created_timer;
struct itimerspec new,old;

timer_create(CLOCKID, _ ,  &created_timer);

new.it_value.tv_sec=Interval.tv_sec;
new.it_value.tv_nsec=Interval.tv_nsec;
new.it_interval.tv_sec=0;// Set interval to 0
new.it_interval.tv_nsec=0;

i=timer_settime(created_timer, 0,&new, &old);
```

[10]